**Customer Purchase Analytics for E-commerce Platform**

**Data Analytics Tasks:**

**A. Basic Aggregation:**

**A. Basic Aggregation**

**Findings:**

* Simple aggregations like SUM, AVG, COUNT, etc., can be used to understand the overall performance of the business.
* These metrics are useful for quickly summarizing data such as total sales or the number of products sold.

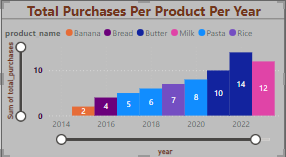
**Insights:**

* Aggregated data provides a high-level view of business performance and can identify broad trends.

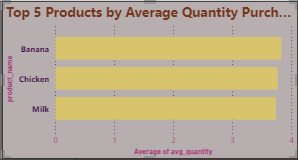
**Recommendations:**

* Use basic aggregation as the starting point for analysis, especially when trying to get an overview of sales or performance metrics.
* Combine aggregation with other techniques to uncover deeper insights.

**Visualization:**



**1A Total number of purchases per product per year**

****

**1B: Top-selling products each year**

**SQL Query:**

**1A : Total number of purchases per product per year**

SELECT

product\_id,

YEAR(purchase\_date) AS year,

COUNT(\*) AS total\_purchases

FROM

purchase\_history

GROUP BY

product\_id, YEAR(purchase\_date);

**1B: Top-selling products each year**

WITH ProductSales AS (

SELECT

product\_id,

YEAR(purchase\_date) AS year,

COUNT(\*) AS total\_purchases

FROM

purchase\_history

GROUP BY

product\_id, YEAR(purchase\_date)

),

TopSellingProducts AS (

SELECT

product\_id,

year,

total\_purchases,

ROW\_NUMBER() OVER (PARTITION BY year ORDER BY total\_purchases DESC) AS row\_num

FROM

ProductSales

)

SELECT

tsp.year,

p.product\_name,

tsp.total\_purchases

FROM

TopSellingProducts tsp

JOIN

products p ON tsp.product\_id = p.product\_id

WHERE

tsp.row\_num = 1 order by tsp.year ASC ;

**B. Join Operations:**

**B. Join Operations**

**Findings:**

* Joins between tables like customers, orders, and products allow us to combine data from different sources.
* Inner, left, and right joins offer flexibility in merging datasets based on the business requirements.

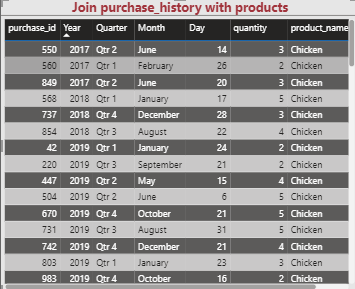
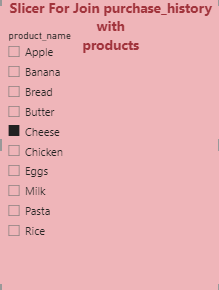
**Insights:**

* Join operations enable a more detailed analysis by linking different business aspects such as customer and product data.

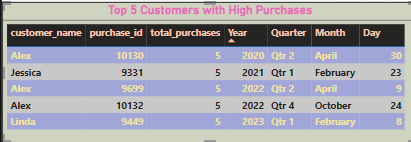
**Recommendations:**

* Use join operations to connect related datasets and gain a comprehensive understanding of customer behavior and sales trends.
* Be cautious with join types to avoid data duplication or loss of critical information.

**Visualization:**

**2A - Average quantity purchased per product**



**2B2 – Top 5 Customers with high purchase**

**SQL Query:**

**2A - Average quantity purchased per product**

SELECT

ph.product\_id,

p.product\_name,

AVG(ph.quantity) AS avg\_quantity

FROM

purchase\_history ph

JOIN

products p ON ph.product\_id = p.product\_id

GROUP BY

ph.product\_id, p.product\_name;

-- Top 5 products by average quantity purchased

WITH AvgQuantity AS (

SELECT

ph.product\_id,

p.product\_name,

AVG(ph.quantity) AS avg\_quantity

FROM

purchase\_history ph

JOIN

products p ON ph.product\_id = p.product\_id

GROUP BY

ph.product\_id, p.product\_name

)

SELECT

product\_name,

avg\_quantity

FROM

AvgQuantity

ORDER BY

avg\_quantity DESC

LIMIT 5;

**2B1:**

SELECT

ph.purchase\_id,

ph.product\_id,

p.product\_name,

ph.purchase\_date,

ph.quantity

FROM

purchase\_history ph

JOIN

products p

ON

ph.product\_id = p.product\_id;

**2B2:**

SELECT

ph.purchase\_id,

ph.customer\_id,

cp.first\_name AS customer\_name,

cp.email AS customer\_email,

ph.purchase\_date,

SUM(ph.quantity) AS total\_purchases

FROM

purchase\_history ph

JOIN

customer\_profile cp

ON

ph.customer\_id = cp.customer\_id

GROUP BY

ph.customer\_id, cp.first\_name, cp.email, ph.purchase\_id, ph.purchase\_date

ORDER BY

total\_purchases DESC

LIMIT 5;

**C. Window Functions:**

**C. Window Functions**

**Findings:**

* Window functions like RANK(), ROW\_NUMBER(), and LEAD() help analyze trends over a partition of data.
* They allow comparison across different subsets without grouping.

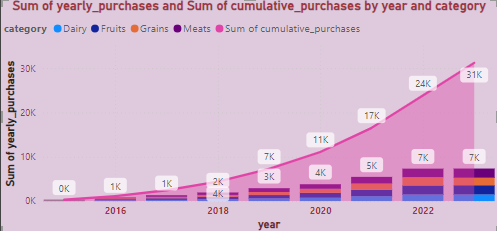
**Insights:**

* Window functions are valuable for trend analysis, ranking products or customers, and identifying outliers.

**Recommendations:**

* Apply window functions when tracking trends or ranking performance metrics, like top customers or popular products, over time.

**Visualization:**

****

**SQL Query:**

SELECT

p.category,

YEAR(ph.purchase\_date) AS year,

SUM(ph.quantity) AS yearly\_purchases,

SUM(SUM(ph.quantity)) OVER (

PARTITION BY p.category

ORDER BY YEAR(ph.purchase\_date)

) AS cumulative\_purchases

FROM

purchase\_history ph

JOIN

products p

ON

ph.product\_id = p.product\_id

GROUP BY

p.category,

YEAR(ph.purchase\_date)

ORDER BY

p.category,

year ASC;

**D. Rank and Dense Rank**

**Findings:**

* Rank and Dense Rank functions are useful for ordering data based on specific metrics, like sales or customer purchases.

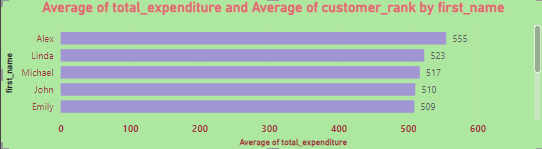
**Insights:**

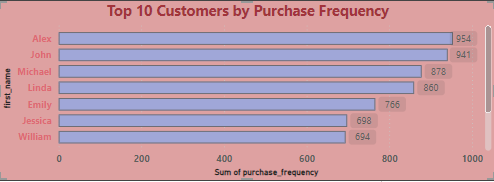
* These functions help in identifying high performers and segmenting them for further analysis.

**Recommendations:**

* Use RANK() for distinguishing between top-performing entities and DENSE\_RANK() for continuous ranking without gaps.

**Visualization:**

****

****

**SQL Query:**

SELECT

customer\_id,

first\_name,

purchase\_frequency,

frequency\_rank

FROM (

SELECT

cp.customer\_id,

cp.first\_name,

COUNT(ph.purchase\_id) AS purchase\_frequency,

DENSE\_RANK() OVER (ORDER BY COUNT(ph.purchase\_id) DESC) AS frequency\_rank

FROM

purchase\_history ph

JOIN

customer\_profile cp ON ph.customer\_id = cp.customer\_id

GROUP BY

cp.customer\_id, cp.first\_name

) ranked\_customers

WHERE

frequency\_rank <= 10

ORDER BY

frequency\_rank;

**E. Percentiles**

**Findings:**

* Percentiles allow you to understand how values are distributed within a dataset, such as identifying the top 10% of customers based on spending.

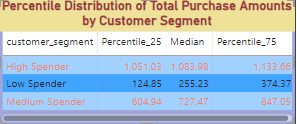
**Insights:**

* Percentiles provide a powerful tool for segmentation and targeting high-value customers.

**Recommendations:**

* Use percentiles to develop targeted marketing strategies by identifying high-value customers or underperforming segments.

**Visualization:**

****

**SQL Query:**

WITH CustomerPurchases AS (

SELECT

cp.customer\_id,

SUM(ph.quantity \* p.price\_per\_unit) AS total\_purchase\_amount

FROM

purchase\_history ph

JOIN

products p ON ph.product\_id = p.product\_id

JOIN

customer\_profile cp ON ph.customer\_id = cp.customer\_id

GROUP BY

cp.customer\_id

),

SegmentedCustomers AS (

SELECT

customer\_id,

total\_purchase\_amount,

CASE

WHEN total\_purchase\_amount >= 1000 THEN 'High Spender'

WHEN total\_purchase\_amount BETWEEN 500 AND 999 THEN 'Medium Spender'

ELSE 'Low Spender'

END AS customer\_segment

FROM

CustomerPurchases

),

PercentileRanks AS (

SELECT

customer\_segment,

total\_purchase\_amount,

PERCENT\_RANK() OVER (PARTITION BY customer\_segment ORDER BY total\_purchase\_amount) AS percentile\_rank

FROM

SegmentedCustomers

)

SELECT

customer\_segment,

MAX(CASE WHEN percentile\_rank <= 0.25 THEN total\_purchase\_amount END) AS Percentile\_25,

MAX(CASE WHEN percentile\_rank <= 0.50 THEN total\_purchase\_amount END) AS Median,

MAX(CASE WHEN percentile\_rank <= 0.75 THEN total\_purchase\_amount END) AS Percentile\_75

FROM

PercentileRanks

GROUP BY

customer\_segment;

**F. Median Calculation**

**Findings:**

* The median provides a better measure of central tendency in skewed datasets, as it is less affected by outliers compared to the mean.

**Insights:**

* The median helps when assessing customer spending habits or product prices that vary widely.

**Recommendations:**

* Use the median to assess the "typical" value in datasets with extreme variations (e.g., salaries, sales prices).

**Visualization:**

****

**SQL Query:**

WITH RankedPurchases AS (

SELECT

p.category,

pu.total\_amount,

ROW\_NUMBER() OVER (PARTITION BY p.category ORDER BY pu.total\_amount) AS RowAsc,

ROW\_NUMBER() OVER (PARTITION BY p.category ORDER BY pu.total\_amount DESC) AS RowDesc

FROM purchases pu

JOIN products p ON pu.product\_id = p.product\_id

JOIN customer\_profile cp ON pu.customer\_id = cp.customer\_id

WHERE p.category IN ('grains', 'meats', 'fruits','Dairy') -- Filter for specific categories

)

SELECT

category,

AVG(total\_amount) AS median\_purchase\_amount

FROM RankedPurchases

WHERE RowAsc = RowDesc

GROUP BY category;

**G. Complex Aggregation**

**Findings:**

* Complex aggregations involve grouping data and applying advanced aggregate functions (like SUM, AVG, COUNT) with specific conditions.

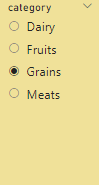
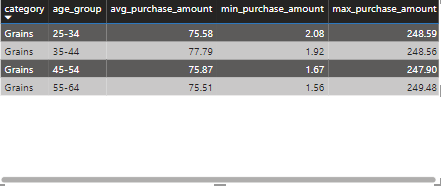
**Insights:**

* Complex aggregation allows deeper analysis, such as revenue per customer or product category.

**Recommendations:**

* Use complex aggregations to break down performance by categories or time periods, especially when analyzing trends or revenue.

**Visualization:**

****

**SQL Query:**

WITH CustomerAgeGroup AS (

SELECT

customer\_id,

CASE

WHEN TIMESTAMPDIFF(YEAR, c.date\_of\_birth, CURDATE()) BETWEEN 18 AND 24 THEN '18-24'

WHEN TIMESTAMPDIFF(YEAR, c.date\_of\_birth, CURDATE()) BETWEEN 25 AND 34 THEN '25-34'

WHEN TIMESTAMPDIFF(YEAR, c.date\_of\_birth, CURDATE()) BETWEEN 35 AND 44 THEN '35-44'

WHEN TIMESTAMPDIFF(YEAR, c.date\_of\_birth, CURDATE()) BETWEEN 45 AND 54 THEN '45-54'

WHEN TIMESTAMPDIFF(YEAR, c.date\_of\_birth, CURDATE()) BETWEEN 55 AND 64 THEN '55-64'

WHEN TIMESTAMPDIFF(YEAR, c.date\_of\_birth, CURDATE()) >= 65 THEN '65+'

ELSE 'Unknown'

END AS age\_group

FROM customer\_profile c

)

SELECT

p.category,

ag.age\_group,

AVG(pu.total\_amount) AS avg\_purchase\_amount,

MAX(pu.total\_amount) AS max\_purchase\_amount,

MIN(pu.total\_amount) AS min\_purchase\_amount

FROM purchases pu

JOIN products p ON pu.product\_id = p.product\_id

JOIN CustomerAgeGroup ag ON pu.customer\_id = ag.customer\_id

GROUP BY

p.category,

ag.age\_group

ORDER BY

p.category, ag.age\_group;

**H. Grouping**

**Findings:**

* Grouping is essential for segmenting data by different categories, like product type, region, or customer demographics.

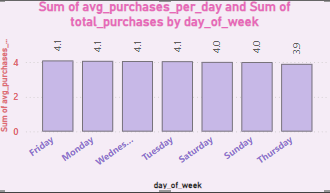
**Insights:**

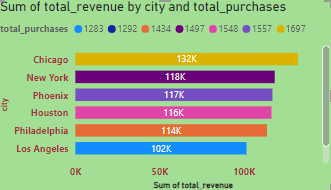
* Grouping enables the comparison of different segments and provides insights into which categories are performing best.

**Recommendations:**

* Use grouping to categorize sales data, customer demographics, or product performance to generate targeted insights.

**Visualization:**

****

****

**SQL Query:**

**H1Grouping**

SELECT

DAYNAME(STR\_TO\_DATE(p.purchase\_date, '%Y-%m-%d')) AS day\_of\_week,

COUNT(\*) AS total\_purchases,

COUNT(\*) / COUNT(DISTINCT STR\_TO\_DATE(p.purchase\_date, '%Y-%m-%d')) AS avg\_purchases\_per\_day

FROM

purchases p

GROUP BY

day\_of\_week

ORDER BY

FIELD(day\_of\_week, 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday');

**H2Grouping:**

SELECT

c.city,

COUNT(p.purchase\_id) AS total\_purchases,

SUM(p.total\_amount) AS total\_revenue

FROM

customer\_profile c

JOIN

purchases p ON c.customer\_id = p.customer\_id

GROUP BY

c.city

ORDER BY

total\_revenue DESC**;**

**I. Case Statement**

**Findings:**

* The CASE statement allows conditional logic in SQL queries to categorize data or calculate values based on specific conditions.

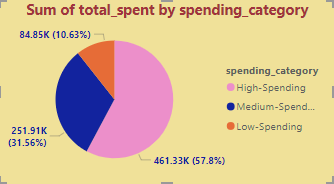
**Insights:**

* This helps in creating custom classifications or handling missing data more effectively.

**Recommendations:**

* Implement the CASE statement to apply custom rules, such as classifying products based on price or determining customer segments based on spending.

**Visualization:**

****

**SQL Query:**

WITH CustomerSpending AS (

SELECT

c.customer\_id,

SUM(p.total\_amount) AS total\_spent

FROM

customer\_profile c

JOIN

purchases p ON c.customer\_id = p.customer\_id

GROUP BY

c.customer\_id

),

RankedSpending AS (

SELECT

customer\_id,

total\_spent,

PERCENT\_RANK() OVER (ORDER BY total\_spent) AS percentile\_rank

FROM

CustomerSpending

)

SELECT

customer\_id,

total\_spent,

CASE

WHEN percentile\_rank <= 0.33 THEN 'Low-Spending'

WHEN percentile\_rank > 0.33 AND percentile\_rank <= 0.66 THEN 'Medium-Spending'

ELSE 'High-Spending'

END AS spending\_category

FROM

RankedSpending

ORDER BY

total\_spent DESC;

**J. Join with Condition**

**Findings:**

* Conditional joins are used to combine datasets based on specific criteria, offering flexibility in how data is merged.

**Insights:**

* This is useful when you need to combine data based on particular conditions like customer behavior or order thresholds.

**Recommendations:**

* Use conditional joins to merge data on specific business rules, like combining customer data with purchase history for targeted insights.

**Visualization:**

****

**SQL Query:**

WITH CustomerWithAge AS (

SELECT

customer\_id,

DATE\_FORMAT(FROM\_DAYS(DATEDIFF(CURDATE(), STR\_TO\_DATE(date\_of\_birth, '%Y-%m-%d'))), '%Y') + 0 AS age

FROM

customer\_profile

)

SELECT

p.purchase\_id,

p.customer\_id,

p.product\_id,

p.purchase\_date,

p.quantity,

p.total\_amount

FROM

purchases p

JOIN

CustomerWithAge cwa ON p.customer\_id = cwa.customer\_id

WHERE

cwa.age > 30;

**K. Top N Analysis**

**Findings:**

* The Top N Analysis allows you to focus on top products, customers, or regions based on a given metric (e.g., total sales, number of purchases).

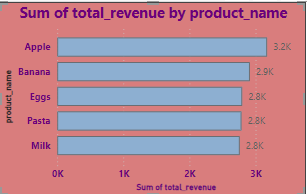
**Insights:**

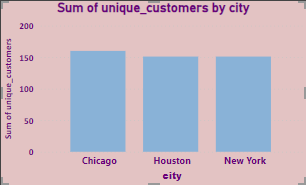
* Identifying top N customers/products helps prioritize business decisions, like inventory management or customer retention.

**Recommendations:**

* Use Top N Analysis for decision-making in areas such as marketing, sales campaigns, and product prioritization.

**Visualization:**

****

****

**SQL Query:**

**K1. Top N Analysis:**

SELECT

pr.product\_name,

p.product\_id,

SUM(p.total\_amount) AS total\_revenue

FROM

purchases p

JOIN

products pr ON p.product\_id = pr.product\_id

GROUP BY

p.product\_id, pr.product\_name

ORDER BY

total\_revenue DESC

LIMIT 5;

**K2. Top N Analysis:**

SELECT

c.city,

COUNT(DISTINCT c.customer\_id) AS unique\_customers

FROM

customer\_profile c

JOIN

purchases p ON c.customer\_id = p.customer\_id

GROUP BY

c.city

ORDER BY

unique\_customers DESC

LIMIT 3;

**L. Window Functions for Trend Analysis**

**Findings:**

* Window functions like LEAD(), LAG(), and CUME\_DIST() can help analyze trends in data across partitions or over time.

**Insights:**

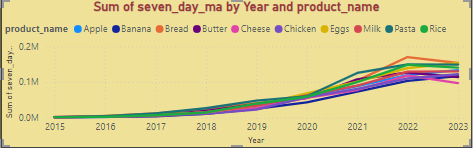
* Window functions enable a deeper understanding of data trends and help with forecasting future performance.

**Recommendations:**

* Apply window functions for predictive analytics, trend analysis, and understanding customer or product lifecycle patterns.

**Visualization:**

****

****

**SQL Query:**

**L. Window Functions for Trend Analysis:**

SELECT

p.product\_id,

pr.product\_name,

p.purchase\_date,

p.customer\_id,

SUM(p.total\_amount) OVER (

PARTITION BY p.product\_id

ORDER BY p.purchase\_date

ROWS BETWEEN 6 PRECEDING AND CURRENT ROW

) AS seven\_day\_ma

FROM

purchases p

JOIN

products pr ON p.product\_id = pr.product\_id

ORDER BY

p.product\_id, p.purchase\_date;

WITH SevenDayMA AS (

SELECT

p.product\_id,

pr.product\_name,

p.purchase\_date,

p.customer\_id,

SUM(p.total\_amount) OVER (

PARTITION BY p.product\_id, p.customer\_id

ORDER BY p.purchase\_date

ROWS BETWEEN 6 PRECEDING AND CURRENT ROW

) AS seven\_day\_ma

FROM

purchases p

JOIN

products pr ON p.product\_id = pr.product\_id

)

SELECT

customer\_id,

MAX(seven\_day\_ma) AS highest\_seven\_day\_ma

FROM

SevenDayMA

GROUP BY

customer\_id

ORDER BY

highest\_seven\_day\_ma DESC

LIMIT 1;

**M. Nested Queries**

**Findings:**

* Nested queries or subqueries allow complex data analysis by embedding one query inside another.

**Insights:**

* This is valuable for filtering data based on aggregate values or conditions that require multi-level analysis.

**Recommendations:**

* Use nested queries when dealing with complex conditions that need to be broken down into substeps.

**SQL Query:**

**N. Date Analysis**

**Findings:**

* Date analysis allows for aggregating and grouping data based on dates, such as sales per month or customer acquisition trends over time.

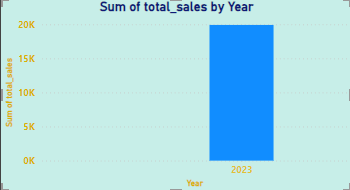
**Insights:**

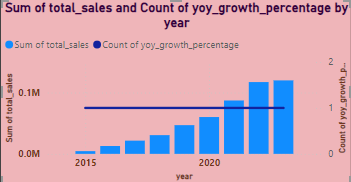
* Date analysis helps businesses track seasonal trends, sales cycles, and growth patterns.

**Recommendations:**

* Use date functions for time-based analysis, like identifying peak sales months or analyzing customer behavior by date.

**Visualization:**

****

****

**SQL Query:**

**N1. Date Analysis:**

SELECT

DATE\_FORMAT(STR\_TO\_DATE(purchase\_date, '%Y-%m-%d'), '%Y-%m') AS sales\_month,

SUM(total\_amount) AS total\_sales

FROM

purchases

GROUP BY

sales\_month

ORDER BY

total\_sales DESC

LIMIT 1;

**N2. Date Analysis:**

WITH yearly\_sales AS (

SELECT

EXTRACT(YEAR FROM pu.purchase\_date) AS year, -- Use the correct column for the purchase date

SUM(pr.price\_per\_unit \* pu.quantity) AS total\_sales

FROM

purchases pu -- Alias for the purchases table

JOIN

products pr ON pu.product\_id = pr.product\_id

GROUP BY

EXTRACT(YEAR FROM pu.purchase\_date) -- Ensure the correct table alias and column

)

SELECT

year,

total\_sales,

LAG(total\_sales) OVER (ORDER BY year) AS previous\_year\_sales,

CASE

WHEN LAG(total\_sales) OVER (ORDER BY year) IS NOT NULL THEN

(total\_sales - LAG(total\_sales) OVER (ORDER BY year))

/ LAG(total\_sales) OVER (ORDER BY year) \* 100

ELSE

NULL

END AS yoy\_growth\_percentage

FROM

yearly\_sales

ORDER BY

year DESC;

**O. Join with Aggregation**

**Findings:**

* Joining data with aggregation helps summarize and combine information, such as total sales per product or average purchase per customer.

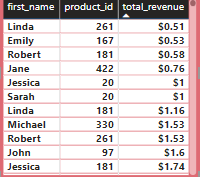
**Insights:**

* It is crucial for understanding performance at a higher level, such as revenue by product or customer demographics.

**Recommendations:**

* Use joins with aggregation to link customer data with sales and product data, providing a comprehensive view of business performance.

**Visualization:**

****

**SQL Query:**

**O. Join with Aggregation:**

SELECT

cp.customer\_id,

cp.first\_name, -- Assuming there's a column for the customer name in customer\_profile

pr.product\_id,

pr.product\_name, -- Assuming there's a column for the product name in products

SUM(pr.price\_per\_unit \* pu.quantity) AS total\_revenue

FROM

customer\_profile cp

JOIN

purchases pu ON cp.customer\_id = pu.customer\_id -- Join customer\_profile with purchases

JOIN

products pr ON pu.product\_id = pr.product\_id -- Join purchases with products

GROUP BY

cp.customer\_id,

cp.first\_name, -- Include customer name if needed in the result

pr.product\_id,

pr.product\_name -- Include product name if needed in the result

ORDER BY

cp.customer\_id,

pr.product\_id;

**P. Customer Retention**

**Findings:**

* Analyzing repeat customers is essential for understanding customer loyalty and measuring retention.

**Insights:**

* Understanding customer retention helps businesses focus on improving loyalty programs and reducing churn.

**Recommendations:**

* Focus on repeat customers by identifying high-value segments and implementing loyalty programs or targeted marketing strategies.

**Visualization:**

****

****

**SQL Query:**

**P1. Customer Retention:**

WITH customer\_purchase\_count AS (

SELECT

customer\_id,

COUNT(DISTINCT purchase\_id) AS purchase\_count

FROM

purchases

GROUP BY

customer\_id

)

SELECT

(COUNT(CASE WHEN purchase\_count > 1 THEN 1 END) \* 100.0) / COUNT(\*) AS repeat\_customer\_percentage

FROM

customer\_purchase\_count;

**P2. Customer Retention:**

WITH customer\_purchase\_dates AS (

SELECT

customer\_id,

MIN(purchase\_date) AS first\_purchase,

MAX(purchase\_date) AS last\_purchase,

DATEDIFF(MAX(purchase\_date), MIN(purchase\_date)) AS days\_between\_purchases

FROM

purchases

GROUP BY

customer\_id

HAVING COUNT(DISTINCT purchase\_id) > 1

)

SELECT

AVG(days\_between\_purchases) AS avg\_days\_between\_purchases

FROM

customer\_purchase\_dates;

**Q. Time Series Analysis**

**Findings:**

* Time series analysis helps in forecasting trends, such as identifying seasonal variations in sales or customer purchases.

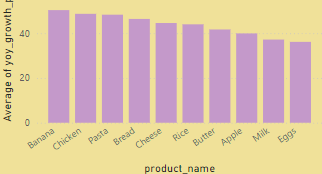
**Insights:**

* Time series analysis allows businesses to predict demand and plan inventory or marketing efforts effectively.

**Recommendations:**

* Use time series analysis for forecasting sales trends, marketing campaign performance, and demand fluctuations.

**Visualization:**

****

**SQL Query:**

**Q. Time Series Analysis:**

WITH product\_purchase\_year AS (

SELECT

p.product\_id,

p.product\_name,

EXTRACT(YEAR FROM pu.purchase\_date) AS purchase\_year,

COUNT(\*) AS purchase\_count

FROM

purchases pu

JOIN

products p ON pu.product\_id = p.product\_id

GROUP BY

p.product\_id, p.product\_name, EXTRACT(YEAR FROM pu.purchase\_date)

),

yearly\_growth AS (

SELECT

current.product\_id,

current.product\_name,

current.purchase\_year,

current.purchase\_count,

COALESCE(

((current.purchase\_count - previous.purchase\_count) / previous.purchase\_count) \* 100,

0

) AS yoy\_growth\_percentage

FROM

product\_purchase\_year current

LEFT JOIN

product\_purchase\_year previous

ON

current.product\_id = previous.product\_id

AND current.purchase\_year = previous.purchase\_year + 1

)

SELECT

product\_id,

product\_name,

purchase\_year,

purchase\_count,

yoy\_growth\_percentage

FROM

yearly\_growth

ORDER BY

yoy\_growth\_percentage DESC;

**R. Subqueries**

**Findings:**

* Subqueries help retrieve data based on specific conditions or filters that cannot be achieved directly in a single query.

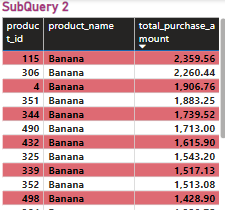
**Insights:**

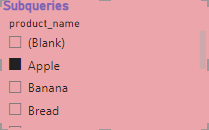
* Subqueries can be powerful for advanced filtering and aggregation when dealing with complex data conditions.

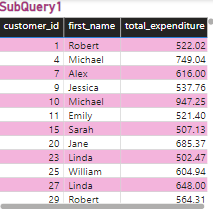
**Recommendations:**

* Use subqueries for scenarios where multiple filtering or aggregation layers are needed.

**Visualization:**

****

****

****

**SQL Query:**

**R1. Subqueries:**

SELECT

c.customer\_id,

c.first\_name,

SUM(p.price\_per\_unit \* pu.quantity) AS total\_expenditure

FROM

customer\_profile c

JOIN

purchases pu ON c.customer\_id = pu.customer\_id

JOIN

products p ON pu.product\_id = p.product\_id

GROUP BY

c.customer\_id, c.first\_name

HAVING

SUM(p.price\_per\_unit \* pu.quantity) > (

SELECT AVG(total\_expenditure)

FROM (

SELECT SUM(p.price\_per\_unit \* pu.quantity) AS total\_expenditure

FROM purchases pu

JOIN products p ON pu.product\_id = p.product\_id

GROUP BY pu.customer\_id

) AS customer\_expenditures

);

**R2. Subqueries**

SELECT

pr.product\_id,

pr.product\_name,

SUM(pr.price\_per\_unit \* pu.quantity) AS total\_purchase\_amount

FROM

products pr

JOIN

purchases pu ON pr.product\_id = pu.product\_id

GROUP BY

pr.product\_id, pr.product\_name

HAVING

SUM(pr.price\_per\_unit \* pu.quantity) > (

SELECT AVG(total\_purchase\_amount)

FROM (

SELECT SUM(pr.price\_per\_unit \* pu.quantity) AS total\_purchase\_amount

FROM purchases pu

JOIN products pr ON pu.product\_id = pr.product\_id

GROUP BY pu.product\_id

) AS product\_purchase\_amounts

);

**S. Correlated Subqueries**

**Findings:**

* Correlated subqueries allow comparison between rows in the outer query and inner subqueries.

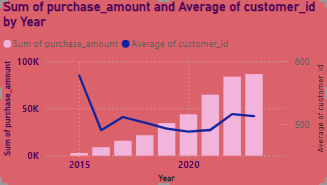
**Insights:**

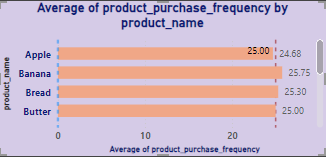
* This type of query is valuable for comparisons across different data subsets, such as identifying customers who purchase above their average spend.

**Recommendations:**

* Use correlated subqueries for advanced filtering and comparison between related rows of data.

**Visualization:**

****

****

**SQL Query:**

**S1. Correlated Subqueries:**

WITH customer\_avg\_purchase AS (

SELECT

pu.customer\_id,

AVG(pr.price\_per\_unit \* pu.quantity) AS avg\_purchase\_amount

FROM

purchases pu

JOIN

products pr ON pu.product\_id = pr.product\_id

GROUP BY

pu.customer\_id

)

SELECT

pu.customer\_id,

pu.purchase\_date,

SUM(pr.price\_per\_unit \* pu.quantity) AS purchase\_amount

FROM

purchases pu

JOIN

products pr ON pu.product\_id = pr.product\_id

GROUP BY

pu.customer\_id, pu.purchase\_date

HAVING

SUM(pr.price\_per\_unit \* pu.quantity) > (

SELECT

avg\_purchase\_amount

FROM

customer\_avg\_purchase cap

WHERE

cap.customer\_id = pu.customer\_id

);

**S2. Correlated Subqueries:**

SELECT

pr.product\_id,

pr.product\_name,

COUNT(pu.purchase\_id) AS product\_purchase\_frequency

FROM

products pr

JOIN

purchases pu ON pr.product\_id = pu.product\_id

GROUP BY

pr.product\_id, pr.product\_name

HAVING

COUNT(pu.purchase\_id) > (

SELECT

AVG(product\_frequency)

FROM (

SELECT

p2.product\_id,

COUNT(pu2.purchase\_id) AS product\_frequency

FROM

products p2

JOIN

purchases pu2 ON p2.product\_id = pu2.product\_id

GROUP BY

p2.product\_id

) AS overall\_avg\_frequency

);

**T. Date Functions**

**Findings:**

* Date functions help extract and manipulate date-related information, enabling time-based analysis.

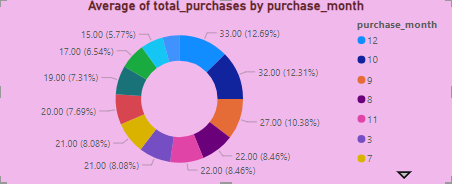
**Insights:**

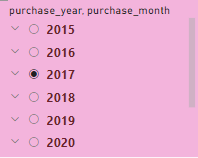
* This allows businesses to segment or aggregate data by year, month, or even day, providing insights into trends over time.

**Recommendations:**

* Use date functions to analyze trends over time, especially when performing time-series analysis or grouping data by date.

**Visualization:**

****

****

**SQL Query:**

**T. Date Functions:**

SELECT

YEAR(p.purchase\_date) AS purchase\_year,

MONTH(p.purchase\_date) AS purchase\_month,

COUNT(p.purchase\_id) AS total\_purchases

FROM

purchases p

GROUP BY

YEAR(p.purchase\_date), MONTH(p.purchase\_date)

ORDER BY

purchase\_year, purchase\_month;

**U. Visualization in Power BI**

**Findings:**

* Power BI offers powerful visualizations like bar charts, line graphs, pie charts, and heat maps for analyzing and presenting data.

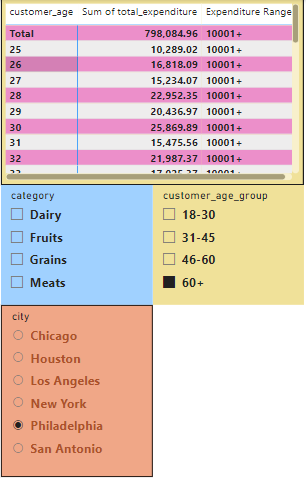
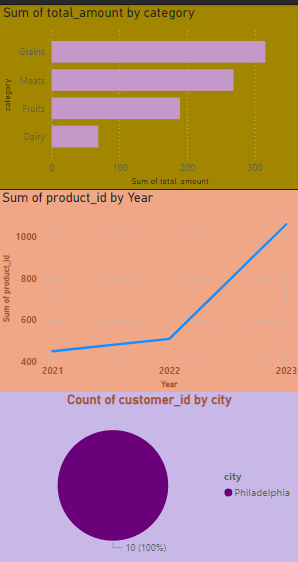
**Insights:**

* Visualizations help to transform complex data into actionable insights for decision-makers.

**Recommendations:**

* Utilize Power BI’s visual capabilities to create interactive dashboards that present key business metrics clearly and effectively.

**Visualization:**

****

**SQL Query:**

**U. Visualization in Power BI:**

SELECT

c.customer\_id,

c.first\_name,

c.last\_name,

c.date\_of\_birth,

p.product\_id,

p.product\_name,

p.category,

pur.purchase\_date,

pur.total\_amount,

c.city,

-- Calculate customer age from date\_of\_birth

TIMESTAMPDIFF(YEAR, c.date\_of\_birth, CURDATE()) AS customer\_age,

-- Age Group categorization based on customer age

CASE

WHEN TIMESTAMPDIFF(YEAR, c.date\_of\_birth, CURDATE()) <= 18 THEN 'Under 18'

WHEN TIMESTAMPDIFF(YEAR, c.date\_of\_birth, CURDATE()) BETWEEN 19 AND 30 THEN '18-30'

WHEN TIMESTAMPDIFF(YEAR, c.date\_of\_birth, CURDATE()) BETWEEN 31 AND 45 THEN '31-45'

WHEN TIMESTAMPDIFF(YEAR, c.date\_of\_birth, CURDATE()) BETWEEN 46 AND 60 THEN '46-60'

ELSE '60+'

END AS customer\_age\_group

FROM

customer\_profile c

JOIN

purchases pur ON c.customer\_id = pur.customer\_id

JOIN

products p ON pur.product\_id = p.product\_id

WHERE

pur.purchase\_date IS NOT NULL;

**V. Advanced Visuals**

**Findings:**

* Power BI supports advanced visuals such as scatter plots, heat maps, and trend lines to uncover patterns and relationships in data.

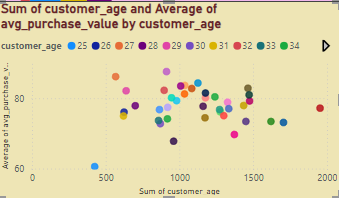
**Insights:**

* These visuals provide deeper insights and allow businesses to track correlations, trends, and patterns across datasets.

**Recommendations:**

* Implement advanced visuals to help in predictive analysis, customer segmentation, and performance tracking.

**Visualization:**

****

**SQL Query:**

**V. Advanced Visuals:**

SELECT

c.customer\_id,

FLOOR(DATEDIFF(CURDATE(), c.date\_of\_birth) / 365) AS customer\_age,

AVG(p.total\_amount) AS avg\_purchase\_value

FROM

customer\_profile c

JOIN

purchases p ON c.customer\_id = p.customer\_id

GROUP BY

c.customer\_id, customer\_age

ORDER BY

customer\_age;

**W. Power BI Dashboard**

**Findings:**

* Combining multiple visualizations into a Power BI dashboard helps summarize key performance indicators (KPIs) for quick decision-making.

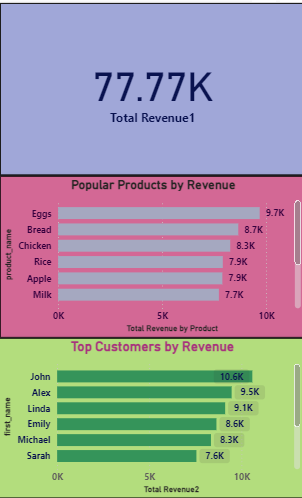
**Insights:**

* Dashboards provide a snapshot of business performance, making it easy to monitor KPIs and make timely decisions.

**Recommendations:**

* Design a dashboard that includes KPIs like total revenue, customer performance, and product sales to offer insights at a glance.

**Visualization:**

****

**SQL Query:**

**W. Power BI Dashboard:**

SELECT

p.product\_id,

p.product\_name,

p.category AS product\_category,

c.customer\_id,

c.first\_name,

c.city AS customer\_city,

FLOOR(DATEDIFF(CURRENT\_DATE, c.date\_of\_birth) / 365) AS customer\_age,

pu.purchase\_date,

pu.total\_amount,

pu.total\_amount AS total\_expenditure -- Alias for simplicity

FROM

customer\_profile c

JOIN

purchases pu ON c.customer\_id = pu.customer\_id

JOIN

products p ON pu.product\_id = p.product\_id

ORDER BY

pu.purchase\_date DESC;

**Overall Dashboard:**

